HAL
open science

Planning with non-deterministic events for enhanced robot autonomy

Jérôme Perret, Rachid Alami

To cite this version:

Jérôme Perret, Rachid Alami. Planning with non-deterministic events for enhanced robot autonomy. Robotics and Autonomous Systems, 1996, 18 (3), pp.311-317. 10.1016/0921-8890(95)00086-0 . hal-01979797

HAL Id: hal-01979797

https://laas.hal.science/hal-01979797

Submitted on 13 Jan 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

PLANNING WITH NON-DETERMINISTIC EVENTS FOR ENHANCED ROBOT AUTONOMY

Jérôme Perret - Rachid Alami*

**LAAS/CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cédex, France*

Abstract. *Beyond pure academic research, a number of application fields raise a challenge. In activities such as planetary exploration, undersea servicing, work in nuclear plants, or disaster intervention, the dynamics of the environment and the strong constraints on data transmissions call for advanced robot autonomy. In this paper we propose a cooperative robot control system architecture based on a reaction planner. We develop our representation for action and event operators and present a simple algorithm for building robust plans. Finally, we discuss plan execution and give hints of possible future developments.*

Key Words. Robotics, Reaction Planning, Autonomous Systems, Plan Execution

1. INTRODUCTION

An autonomous robot system operating in an environment in which there is uncertainty and change needs to combine reasoning with reaction. Reasoning means the ability to decide what to do; the reasoning activity we are interested in here is planning, as the action of projecting the current situation into the future in order to figure out a sequence of actions leading to a goal. Reaction means the ability to act in order to avoid the negative consequences of a situation, or on the contrary to benefit from its positive effects.

The choice of the "good" reaction is usually not trivial, and might require planning. However, in the case of an automatic planning system, the time needed to produce a plan can be very long, which contradicts the notion of "reaction" itself.

In this context, several approaches have been proposed, which give a partial solution to the problem. We will distinguish here three main strategies:

Anytime planning: (Dean and Boddy, 1988) Anytime algorithms have the ability to produce a plan within a given bounded time compatible with the necessary reaction. This approach tries to bring reasoning nearer to reactivity. The price to pay for this is the poor quality of the plan when the time allocated to planning is short. Moreover, in order to build a complete control system, there is a need for a meta-supervision level in charge of deciding when to plan and how much time to allocate to planning in a given situation.

Probabilistic planning: (Kushmeric *et al.*, 1993) The purpose of this approach is to plan for a sequence of actions which, whatever the initial state of the environment and whatever its evolution, will lead to the goal with a sufficiently high probability. In that way, uncertainty in the environment is accounted for, and the control system does not have to rely heavily on the poor perception capabilities of the robot. The problem is the existence of such plans, and the complexity of their construction.

Reaction planning: (Thiébaux and Hertzberg, 1992) Reaction planning means to foresee the changes in the environment and to produce reactions in advance. The strong hypothesis in this approach is the complete knowledge of the world and its possible evolution, and of the consequences of the robot's actions.

Finally, a number of recent works try to bring these approaches together and associate them (Drummond and Bresina, 1990). However, they hit the full complexity of the problem and are forced to make compromises.

Our claim is that reasoning and reaction are two activities which must remain distinct, and cooperate. In that way, we definitively place ourselves in the third strategy, reaction planning. The key point is the interaction between the two processes, and previous works defer largely there.

In section 2, we will discuss this interaction, and establish what conditions it imposes on produced plans. In section 3 we will describe the class of problems we are interested in here. In section 4 we

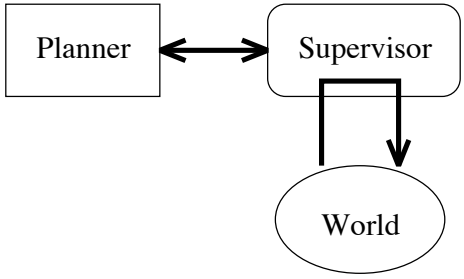

Figure 1 The Planner-Supervisor paradigm

will present our planning operators and in section 5 we will propose a planning scheme for producing plans which satisfy the conditions introduced in section 2. Section 6 will deal with plan execution. Finally, we will give an overview of our future work and a short conclusion.

2. DISCUSSION

Our system design is simple and now rather commonplace (see fig. 1). Several authors use the same description, such as (Fikes *et al.*, 1972), (Drummond and Bresina, 1990), (Kabanza, 1992), (Musliner *et al.*, 1992).

The responsibility of "closing the loop" at the level of plan execution control is entirely devoted to the supervisor. In order to achieve it, the supervisor makes use only of deliberation algorithms which are guaranteed to be time-bounded and compatible with the dynamics of the controlled system. Indeed, all deliberation algorithms which do not verify this property are actually performed by the planner upon request of the supervisor.

We are primarily interested here in the planning module, so we will develop the conditions which the plans have to fulfil, and we will come back to the supervision at the end of the article.

We say that, in order to insure the completion of the task and the safety of the robot, the plans have to be: safe, complete, and goal-achieving. We call such a plan **robust**.

Safe means it shouldn't lead the robot into a dangerous situation and leave it there without a handy reaction. Complete means all courses of events are accounted for. Goal-achieving means the plan leads to the goal.

Given our strong hypotheses and the complexity, the task of building a robust plan is way beyond our reach. Thus we adopt a **moderated** definition of robustness, based on the three following rules:

1. safety: while applying the plan to the known initial state of the world, given its foreseeable evolution, no situation shall arise where the *safety condition* is unsatisfied;
2. completeness: every foreseeable course of events shall be accounted for, i.e. followed by an action **or a** *replan* **statement**; no *replan* statement shall take care of a situation which could foreseeably become dangerous while replanning (replanning can take an unknown unbounded time). This is a key point in our approach, which helps us limit the exploration, and thus envisage to address more ambitious problems;
3. goal-achieving: the probability of reaching the goal by applying the plan to the known initial state of the world, given its foreseeable evolution, shall be non-null (and superior to a given threshold);

Several previous works give partial answers to theses rules:

(Kabanza, 1992; Godefroid and Kabanza, 1991): F. Kabanza proposes a search algorithm in a graph of states which transitions correspond to actions (operators which the system can control) or events (operators over which the system has no control); his system uses the search itself for generating *liveness rules* (leading towards the goal), *safety rules* (leading away from danger), and *heuristic rules* (which can be used at runtime to deal with an unforeseen situation). Efficient as it may be for a specific class of problems, the search in a graph of states hits the full combinatorial complexity in any closer-to-reality world. Moreover, Kabanza cannot avoid to manage the conflicts between generated rules, which can turn out to be very dangerous in the end.

(Musliner *et al.*, 1992): CIRCA is a complete integration example for a robot control system; by reasoning on a *graph model of RTS/environment interaction*, it generates reactions which are necessary to achieve the goals; from these reactions, it computes a *cyclic schedule* to be executed by the RTS. However, the graph model representation appears to be very complex and difficult to expand to a realistic world domain; this compels CIRCA to remain a low-level control system, which actually is its primary purpose.

(Thiébaux and Hertzberg, 1992): S. Thiébaux and J. Hertzberg propose a non-deterministic action model and a planning method which produces plans as *T-M graphs*; these plans can in turn be translated into finite state automata, which makes them executable and enlarges their applicability scope. However, they

avoid modelling foreseeable changes in the environment, which are indeed partly responsible of the non-determinism of the actions.

3. OUR CLASS OF PROBLEMS

Our domain is mobile robots in changing environments. We define a class of problems characterized as follows:

- the robot system has an exact knowledge of the characteristics of its environment relevant to its task, or it can find them out by executing a specific action;
- these characteristics include the state of the environment and its *reactivity*, i.e. its foreseeable evolution in the context of robot activity;
- moreover, it has an exact model of the actions it can execute; this model includes their non-deterministic consequences and their probability.

Following are two example worlds which belong to this class of problems:

First example: The robot's task is to enter a building in which a chemical incident occurred; it carries a spray which can neutralize one of the contaminating chemicals (but not all that may have been spilled); it can be itself contaminated if in presence of a corrosive chemical for too long, and the contamination is fatal in the short term; it can decontaminate itself by going out of the building and into a cleaning area.

Second example: The robot's task is to carry objects around inside a factory; the production process implies strong unscheduled constraints on its movements; moreover, it is dangerous to stay at some crossings because of heavy carriers.

We will develop the first example in the following sections.

4. REPRESENTATION

Our planning operators are variants of STRIPS operators with pre and postconditions. As in (Thiébaux and Hertzberg, 1992) and (Kushmeric *et al.*, 1993), we allow for alternative outcomes of actions applied in the same situation, and to every set of consequences of the application of an operator, we associate a probability.

We distinguish between *actions* and *events*. Actions are operators over which the robot system has full control, therefore they are akin to the STRIPS planning operators. On the other hand, events are operators over which the robot system has no control; the preconditions of an event are its trigger, and the postconditions are its consequences. The total probability of all outcomes of an action has to be 1, whereas for an event it can be lower (meaning that an event may or may not occur). Figures 2 and 3 present some sample operators, actions and events. Please note that event `robot-dead-contaminated` violates the safety condition `(robot-safe)`, which no action can re-assert.

```
(def-action check-for-room-contamination
  :args (?room)
  :precond (
    (room ?room)
    (robot-at ?room))
  :effects (
    (with-probability
      (0.8 ((:add (room-clean ?room))))
      (0.2
        ((:add (room-contaminated ?room))))))
  :duration 1)
```

Figure 2 *A sample action*

```
(def-event robot-contaminated
  :vars (?room)
  :when (
    (robot-at ?room)
    (room-contaminated ?room))
  :effects (
    (with-probability
      (0.5 ((:add (robot-contaminated))))))
  :delay 10)

(def-event robot-dead-contaminated
  :vars ()
  :when (
    (robot-contaminated))
  :effects (
    (with-probability
      (1.  ((:add (robot-dead))
            (:del (robot-safe))))))
  :delay 25)
```

Figure 3 *Two sample events*

Action operators can be used to describe robot actions, reactions of the environment to robot actions (e.g. while spraying decontaminants, a smoke alarm could be raised automatically), and actions of the lower robot control layers (e.g. when moving along a corridor, the robot might stop because of an obstacle). Event operators account

```
(defstep
    :action (check-for-room-contamination ?room)
    :precond '((room ?room)(robot-at ?room))
    :add '((room-clean ?room))
    :dele nil
    :equals nil)

(defstep
    :action (check-for-room-contamination ?room)
    :precond '((room ?room)(robot-at ?room))
    :add '((room-contaminated ?room))
    :dele nil
    :equals nil)
```

Figure 4 *Generated SNLP operators*

for the evolution of the world state only (e.g. the light being turned off by the time switch, the robot becoming contaminated).

The planning problem is defined as follows: given the initial state of the world (a set of predicates) and the goal, actions and events as described above, the *safety condition* not to be violated, the *replan threshold* and the *goal achievement threshold*, find a **robust plan** (i.e. satisfying the conditions given in section 2). It may be that the goal can be achieved in a situation which is not stable (in which events can occur); in that case, the planner should continue to plan until it has reached a stable state. The concept of *stability* is fundamental in our approach; we assume here that any stable situation is suitable for replanning. In that way, the planner can insure that it will be called again eventually, possibly with a new task to perform.

5. SIMPLE PLANNING ALGORITHM

We propose a first simple planning algorithm, much in the fashion of Warren's *WARPLAN-C* (Warren, 1976). It is based on the deterministic non-linear planner SNLP (Mc Allester and Rosenblitt, 1991; Barrett and Weld, 1992), but could be applied to other deterministic planners

The action operators are first compiled into deterministic operators, one operator being created for every possible outcome of an action. Figure 4 presents the operators for the `check-for-room-contamination` action.

Given the initial state of the world and the compiled operators, a first linear plan is build by the deterministic planner, consisting of a sequence of deterministic operators. Now, using the original actions and events, this plan is expanded into a tree structure akin to the T-M graphs of (Thiébaux and Hertzberg, 1992); this structure is composed of three types of nodes: A (i.e. start of an action), E (i.e. event) and S (i.e. world state), and has the following properties:

- the root is a S-node, as are all leaves,
- A-nodes and E-nodes have only one S-node as successor,
- non-leaf S-nodes have either one A-node or a number of E-nodes as successor.

The procedure for building the A-E-S tree is the following: the first action of the linearized deterministic plan is expanded into one A-node (the start of the action) followed by one S-node (its execution state), itself followed by a number of E-nodes and corresponding S-nodes (one pair for each possible outcome of the action); then the execution S-node is tested for events, as well as all other new S-nodes which do not fit in the deterministic plan; for each S-node, all triggered events are sorted using their *delay* parameter, and tested for exclusiveness (i.e. incompatibility of effects); the list is pruned after the first event with a probability of 1 (if present), and one E-node is generated for every remaining (set of) event(s), along with its successor S-node and its probability of occurrence. The root S-node is built from the initial world state, and the procedure is carried on with every S-node along the deterministic plan, until all actions have been translated into A-nodes and no event is applicable in any leaf S-node. The resulting structure may or may not lead to the goal, depending on the triggered events.

For example, suppose we have two actions A and B, and two events E_1 and E_2. A has two possible outcomes, and B only one, leading to three deterministic operators: A_1, A_2, and B_1. Starting from an initial state S_i, the deterministic planner produces the following linear plan: A_2 then B_1. No events are applicable in S_i, so an A-node is built with the start of action A, then a S-node S_A for its execution, leading to two E-nodes, E_{A1} and E_{A2} and their successor S-nodes S_{A1} and S_{A2} (corresponding to outcomes A_1 and A_2 respectively). Event E_1 is applicable in S_{A1}, so we add an E-node E_{E1} and a S-node S_{E1}. From S_{A2} we expand action B, leading to S_B, E_{B1} and S_{B1}. Event E_2 is applicable in state S_B, so we add two nodes: E_{E2} and S_{E2}, leading to the tree structure presented fig. 5.

Now, all new S-nodes are tested for the *safety condition* (e.g. in our example, `(robot-safe)`). Should this condition be false in state S, then we have to plan for a reaction avoiding S. If S's father is an A-node A, then we prune the plan before

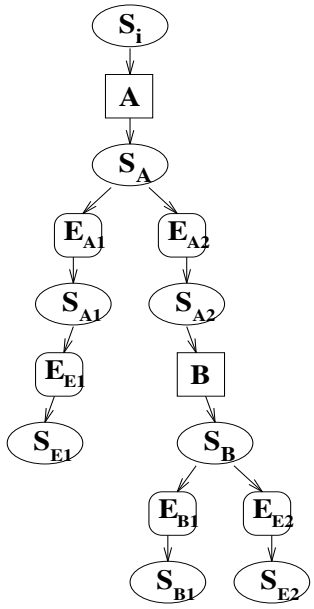

Figure 5 A-E-S tree example

A and look for an other linear plan starting from the new leave. If S's father is an E-node E, again we call the deterministic planner in order to find a new plan, which first action will inhibit E. There are several means to find a new plan: either by using the negation of one of the preconditions of E as new goal, or by inserting any applicable action compatible with the delay of E and planning again from there on. If no plan is found, then the problem is declared unsolved.

If all new S-nodes are safe, then a new S-node with a probability above the *replan threshold* is chosen, among all S-nodes which have no successor A-node. The S-node becomes the initial state, and the deterministic planner is used to build a new plan towards the goal.

In the end, all pending S-nodes which are below the *replan threshold* are flagged with the *replan* statement. The resulting plan is solution if the sum of the probabilities of all goal states is above the *goal achievement threshold*. Figure 6 gives an example of a plan after the first planning step, and at the end of the refinement phase.

Why is our algorithm sound? We can demonstrate that, provided no event loop with a 1 probability exists (which can be easily checked for), the probability keeps decreasing along all branches of the A-E-S tree, and thus will finally go under the replan threshold. Furthermore, the plans produced satisfy the three conditions given in section 2.

We have built a first prototype planner based on SNLP, which managed to produce the plan presented fig. 6. However, it is not complete, as it is not guaranteed to find a plan if one exists. Suppose we add an event stating that the doors close automatically after the robot got through them, and the only way to open them is with a remote-control. After building the first SNLP plan, the system will come up with the event `(door-closed room1)`, and is not going to find a solution because it should have taken the remote-control before entering the room. In order to solve this type of problems, we need the ability to backtrack over goals, which is not yet possible with our prototype.

6. PLAN EXECUTION

The *Supervisor* interacts with the environment and with the planner. The environment (which may include one or more lower control layers) is viewed as a set of processes which exchange signals with the supervisor. These processes correspond to the actions of the agent as well as events associated with environment changes independent from robot actions (Alami *et al.*, 1993).

These processes are under the control of the supervisor which has to comply with their specific features. For example, a process representing a robot motion, cannot be cancelled instantaneously by the supervisor: indeed, such a process has an "inertia". The supervisor may request a stop at any moment during execution; however, the process will go through a series of steps before actually finishing.

The simplest way to represent such processes are finite state automata (FSA). More elaborate representations such as temporized processes should be investigated.

The plan itself can also be understood as a finite automaton: A-nodes and S-nodes correspond to the states of the automaton, A-node and E-nodes to transitions.

In the FSA class we use, at any moment:

- the set of allowed external signals correspond to all the actions that can be taken by the supervisor (either part of the plan or decided by the supervisor's own bounded-time decisional abilities);
- similarly, the set of possible internal signals correspond to all action terminations and results and all environment changes that could be perceived by the supervisor.

The activity of the supervisor consists in moni-

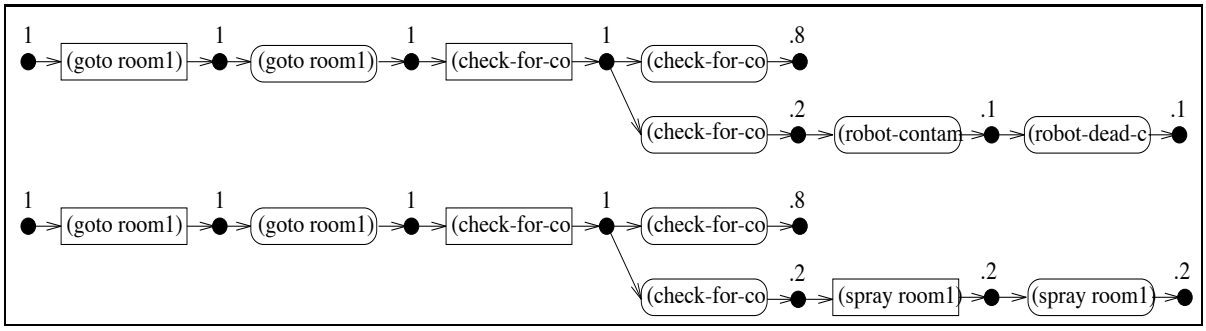

Figure 6 *Two steps of the planning algorithm*

toring the plan execution by performing situation detection and assessment and by taking appropriate decisions in real time, i.e., within time bounds compatible with the rate of the signals produced by the processes and their possible consequences. Such bounded-time decisional abilities are well in the reach of a system like KHEOPS (Ghallab, 1984).

Indeed, every single decision process which can be left to the supervisor, provided it can be done in bounded time, reduces the complexity of the planning problem. For example, suppose the supervisor is able to make the robot leave the building as soon as it becomes contaminated, by following the same path backwards. We could take advantage of this behaviour at planning time by adding a meta-planning rule which takes care of all S-nodes where `(robot-contaminated)` is true and the robot is not too far away from the entrance. That way, we would leave the world model unchanged so that the planner can still plan for these situations if it decides to.

During execution, the robot system will follow one path in the plan automaton, thus validating states which had only a probability of occurrence before. Now with each execution step, we can propagate this validation, dividing the probability of all descendants of the selected state by its own probability, and setting all other states to null. Doing this, it is possible that some states flagged with `(replan)` will raise above the probability threshold and thus become eligible for further planning. In order to save time and profit from our cooperating architecture, the execution supervisor will send right away the most probable state to the planner for further plan refinement.

7. FUTURE WORK

Our first activity will be to implement backtracking strategies in the simple planner and to investigate the possibility of enriching our representation with meta-planning rules. Then we will focus on the supervisor module.

Once the complete system is operational, we will test it extensively on different problems, trying to evaluate the robustness of produced plans and in particular the stability regarding modelling errors (action outcomes' and events' probabilities are especially difficult to estimate).

We hope to give some interesting results of these activities in the final presentation at IAS-4.

8. CONCLUSION

After having justified our approach to integrating reasoning and reaction in the context of autonomous mobile robots, we have described the planner/supervisor interaction and have derived the key properties of the plans to be produced. Then, we have proposed a model for actions and events and a tentative planning algorithm for generating such plans, which we have demonstrated on an example. Finally, we have proposed a plan execution scheme, addressing the issues of planner/supervisor interaction and real-time decision-making.

REFERENCES

Alami, R., Chatila, R. and Espiau, B. (1993). Designing an Intelligent Control Architecture for Autonomous Robots. In: *ICAR '93*. Tokyo, Japan. pp. 435–440.

Barrett, A. and Weld, D. (1992). Partial Order Planning: Evaluating Possible Efficiency Gains. Technical Report 92-05-01. Univ. of Washington, Dept. of Computer Science and Engineering.

Dean, T. and Boddy, M. (1988). An Analysis of Time-Dependent Planning. In: *AAAI-88*. pp. 49–54.

Drummond, M. and Bresina, J. (1990). Anytime Synthetic Projection: Maximizing the Probability of Goal Satisfaction. In: *AAAI-90*. Boston. pp. 138–144.

Fikes, R. E., Hart, P. E. and Nilsson, N. J. (1972). Learning and Executing Generalized Plans. *Artificial Intelligence* **3**(4), 251–288.

Ghallab, M. (1984). Task execution Monitoring by compiled production rules in an advanced multi-sensor robot. In: *Robotics Research : The Second International Symposium, Kyoto (Japan)*.

Godefroid, P. and Kabanza, F. (1991). An Efficient Reactive Planner for Synthesizing Reactive Plans. In: *AAAI-91*. pp. 640–645.

Kabanza, F. (1992). Reactive Planning of Immediate Actions. PhD thesis. Université de Liège.

Kushmeric, N., Hanks, S. and Weld, D. (1993). An Algorithm for Probabilistic Planning. Technical Report 93-06-03. Univ. of Washington, Dept. of Computer Science and Engineering. To appear in *Artificial Intelligence*.

Mc Allester, D. and Rosenblitt, D. (1991). Systematic Nonlinear Planning. In: *AAAI-91*. pp. 634–639.

Musliner, D. J., Durfee, E. H. and Shin, K. G. (1992). CIRCA: A Cooperative Intelligent Real-Time Control Architecture. *IEEE Transactions on Systems, Man, and Cybernetix*.

Thiébaux, S. and Hertzberg, J. (1992). A Semi-Reactive Planner Based on a Possible Models Action Formalization. In: *1st AIPS*. pp. 228–235.

Warren, D. H. D. (1976). Generating Conditional Plans and Programs. In: *AISB-76 Summer Conference*. Edinburgh.